本章介绍

<$在线程之间划分数据的技术

<$影响并发代码性能的因素

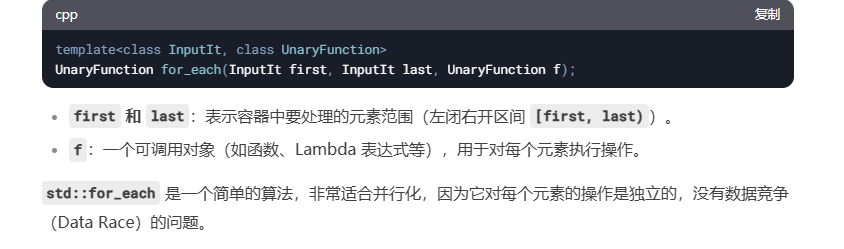
<$性能因素如何影响数据结构的设计

<$多线程代码中的异常安全性

<$可伸缩性

<$几种并行算法的示例实现

对于线程，同样的问题也适用。您需要决定使用多少线程以及它们应该执行哪些任务。您需要决定是让“通用”线程在任何时间点完成任何必要的工作，还是让“专业”线程做好一件事，或者两者兼而有之。无论使用并发的驱动原因是什么，您都需要做出这些选择，而您如何做这些选择将对代码的性能和清晰度产生关键影响。

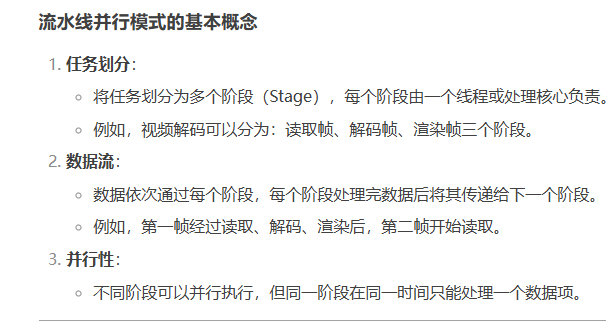


偶尔，其他线程可能会给它需要处理的数据或触发事件，但通常每个线程都专注于做好一件事。就其本身而言，这是基本的良好设计;每段代码都应该有一个单一的责任。



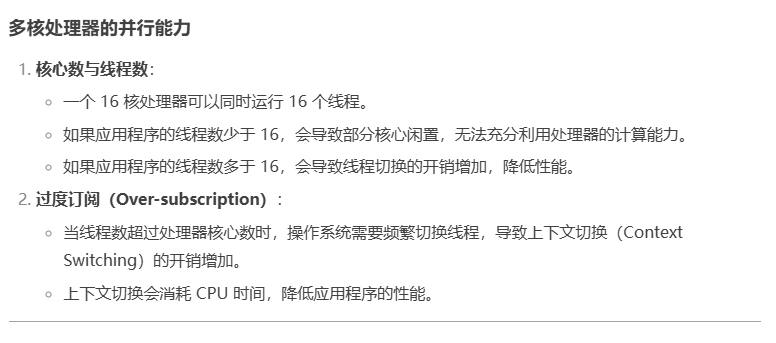
文本

描述已自动生成





现在让我们开门见山地看看其中的一些因素，从最明显的一个开始：您的目标系统有多少个处理器？



图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

如果一个处理器因为必须等待缓存传输而停顿，那么它在此期间不能做任何工作，即使有其他线程在等待，可以做有用的工作，所以这对整个应用程序来说是个坏消息。

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

文本, 信件

描述已自动生成

文本

描述已自动生成

图形用户界面, 文本, 信件, 电子邮件

描述已自动生成

由同一线程访问的数据在存储器中靠近在一起（并且因此更可能在同一高速缓存行中



尝试调整线程之间的数据分布，以便同一线程处理靠近的数据。

尽量减少任何给定线程所需的数据。尝试确保不同线程访问的数据相距足够远，以避免使用std：：hardware\_destructive\_interference\_size作为指导的错误共享。

